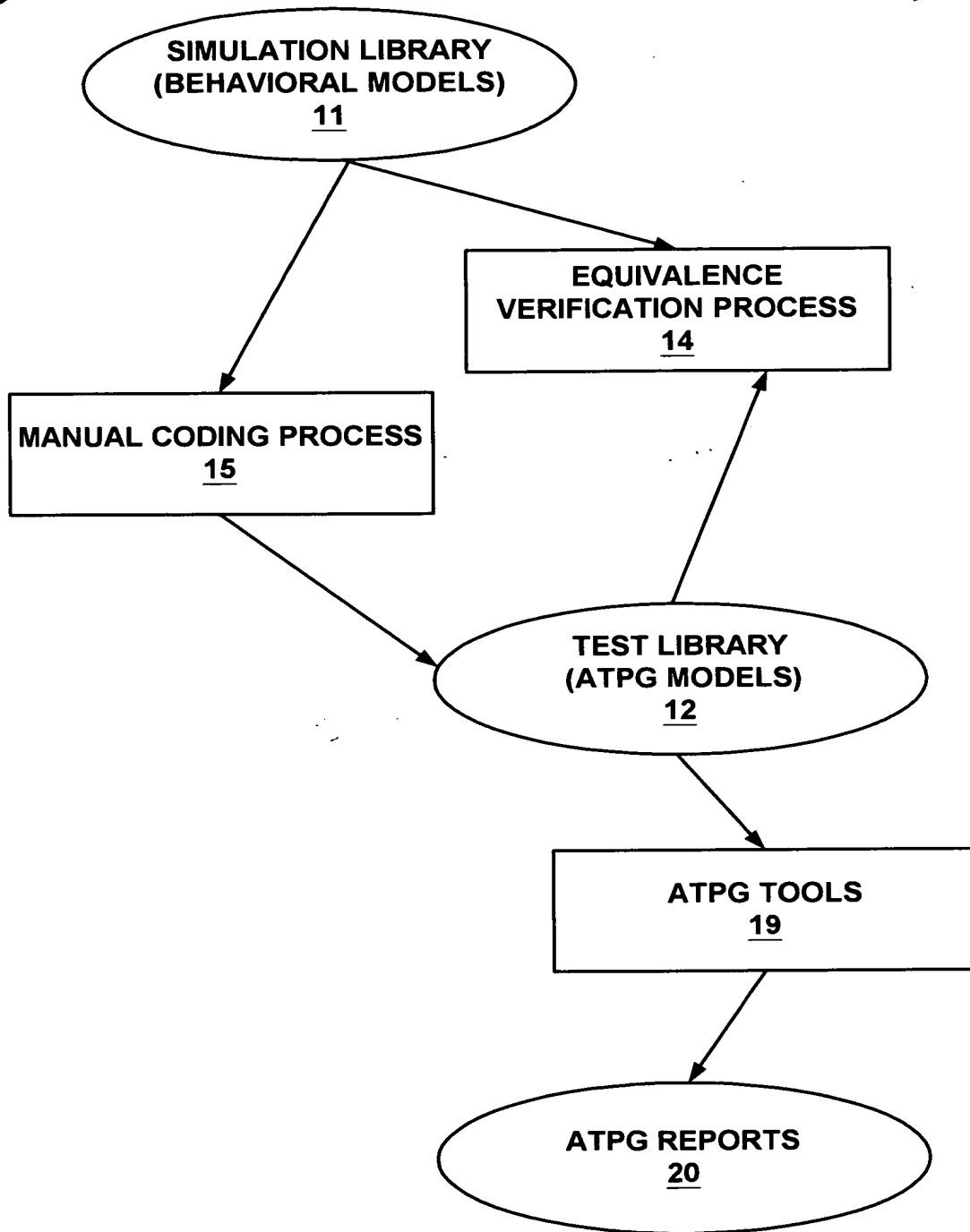
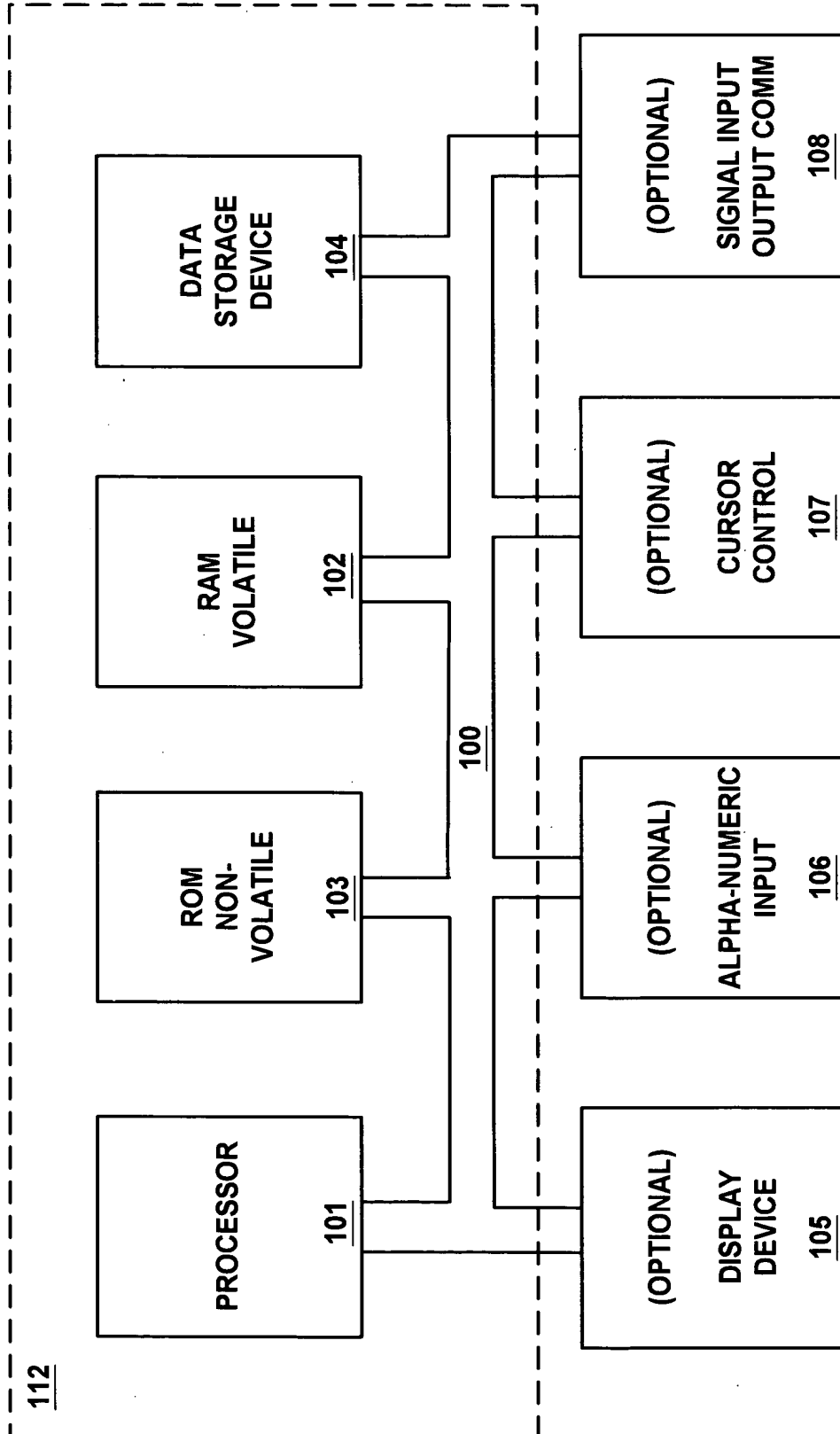




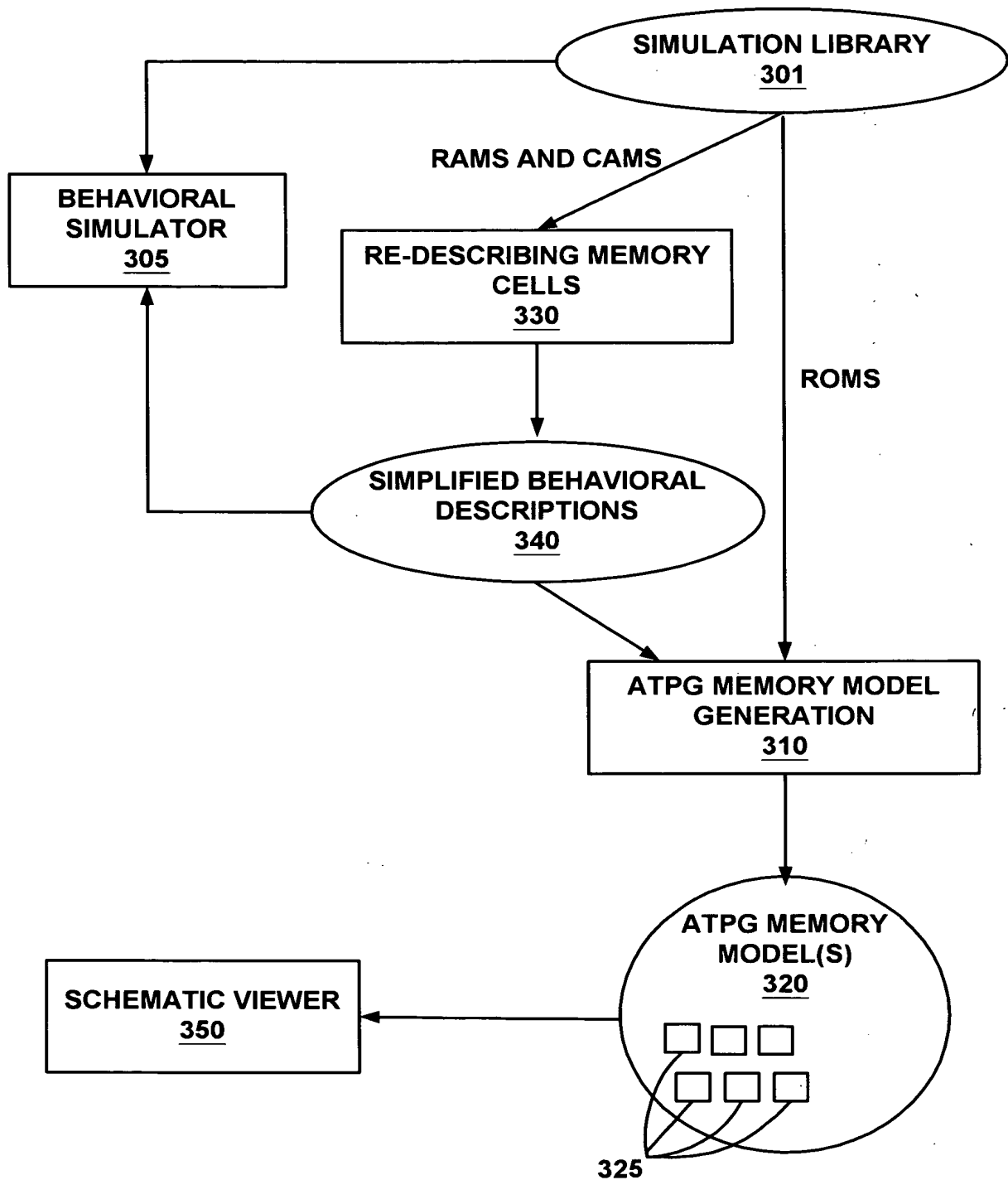
1/22



**FIGURE 1  
(PRIOR ART)**



**FIGURE 2**

300**FIGURE 3**

```

memory_port :: = read_port | write_port | set_port | reset_port
read_port :: = always @ ( level_read_port | edge_read_port )
write_port :: = always @ ( level_write_port | edge_write_port )
reset_port :: = [begin] sr_control memory_name [address_nef] <= 0_constant; [signal]
set_port :: = [begin] sr_control memory_name [address_nef] <= 1_constant; [signal]
level_read_port :: = [ ( ) address_net [or control_nef] [or eventf] [ ] ]
    (net_control | addr_control) read_port_assign [ else reg_net <= constant ; ]
level_write_port :: = ( address_net or control_net or data_net )
    net_control write_port_assign
edge_read_port :: = (edge clock_net) (net_control | addr_control) read_port_assign
edge_write_port :: = (edge clock_net) [net_control] write_port_assign
read_port_assign :: = reg_net <= memory_name [ address_net ] ;
write_port_assign :: = [begin] memory_name [ address_net ] <= data_net ; [signal]
sr_control :: = always @ control_net net_control
    for ( index = 0 ; index < max_address; index = index + 1 )
net_control :: = if ( [!] control_net )
addr_control :: = if ( address_net <= max_address )
signal :: = #0; -> event ; end
constant :: = number_bits ` base { digit } +
edge :: = posedge | negedge
base :: = b | B | h | H
digit :: = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | a | A | b | B | c | d | D | e | E | f | F | x | X | z | Z

```

FIGURE 4

```

`celldefine
`ifdef verifault
    `suppress_faults
    `enable_portfaults
`endif
module ROM ( ... );
output ...; input ...; wire ...;
reg [3:0] _dout;
reg [3:0] rom_data [ 0:127 ];
    // control circuit
    buf ( _H15, H15 );
    ....
    //data out
    buf ( N01,_dout [0] );
    ....
    //address
    buf ( _H07, H07 );
    buf ( _H14, H14 );
    and ( _a [6], _H07, _H15 );
    and ( _ta [6], _H14, _TEB );
    or ( _ad [6], _a [6], _ta [6] );
initial
    $readmemh ("file" , rom_data );
always @_ad // read port
    if ( _ad  <= 127 )
        _dout = rom_data [_ad];
    else
        _dout = 4'bx;
specify
...
endspecify
endmodule
`ifdef verifault
    `nosupress_faults
    `disable_portfaults
`endif
`endcelldefine

```

**FIGURE 5**  
**(CONVENTIONAL ART)**

600

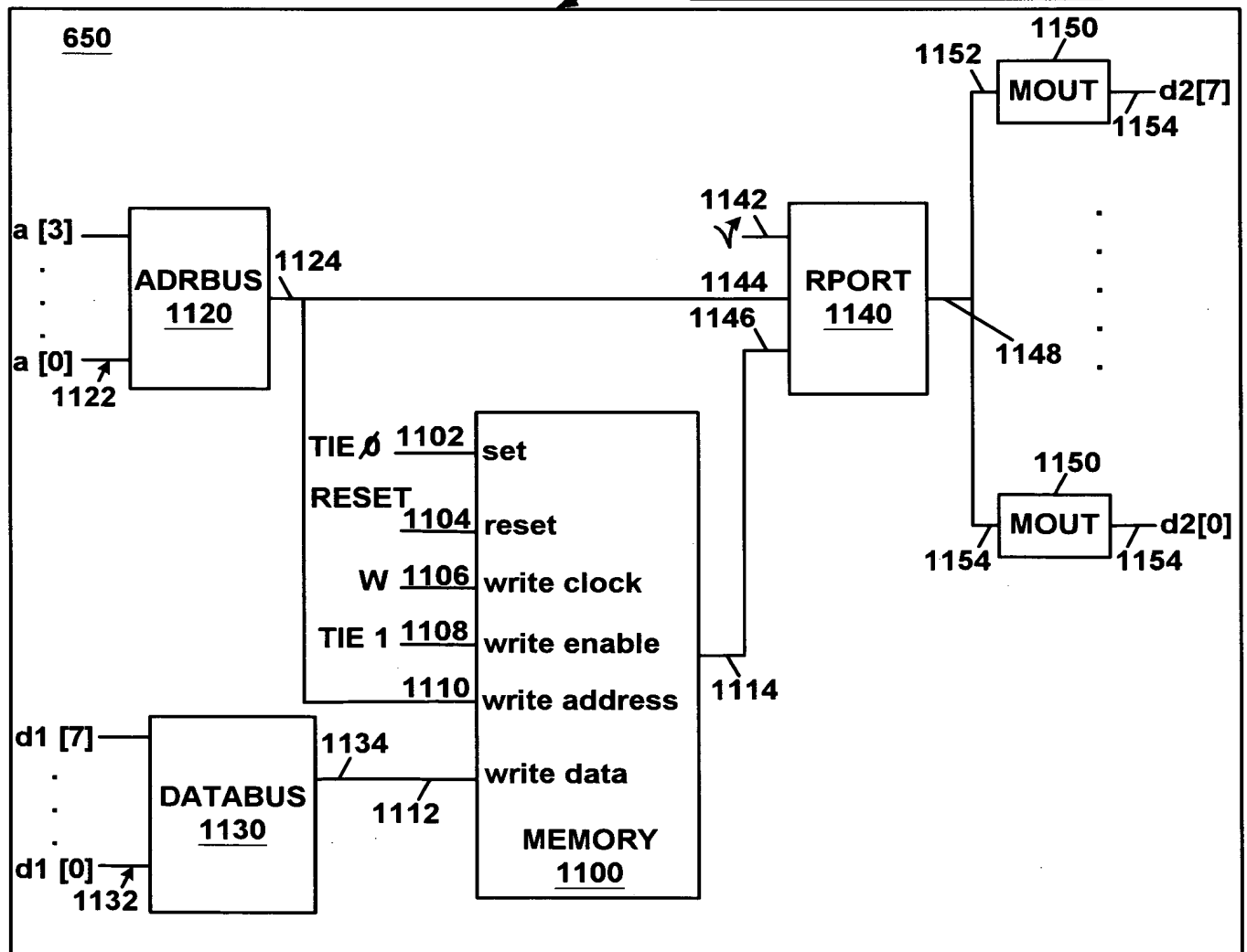
6/22

```

module withram (reset, r,w, a, d1, d2 );
input reset, r,w;
input [3:0] a;
input [7:0] d1;
output [7:0] d2;
    reg [7:0] mymem [15:0], d2;
    integer i;
    always @ reset if (reset)
    for (i=0; i<16; i = i + 1) mymem [i] <= 0;
    always @ (posedge w) mymem [a] <= d1;
    always @ r if (r) d2 <= mymem [a];
    else d2 <= 0; // read_off is 0
endmodule

```

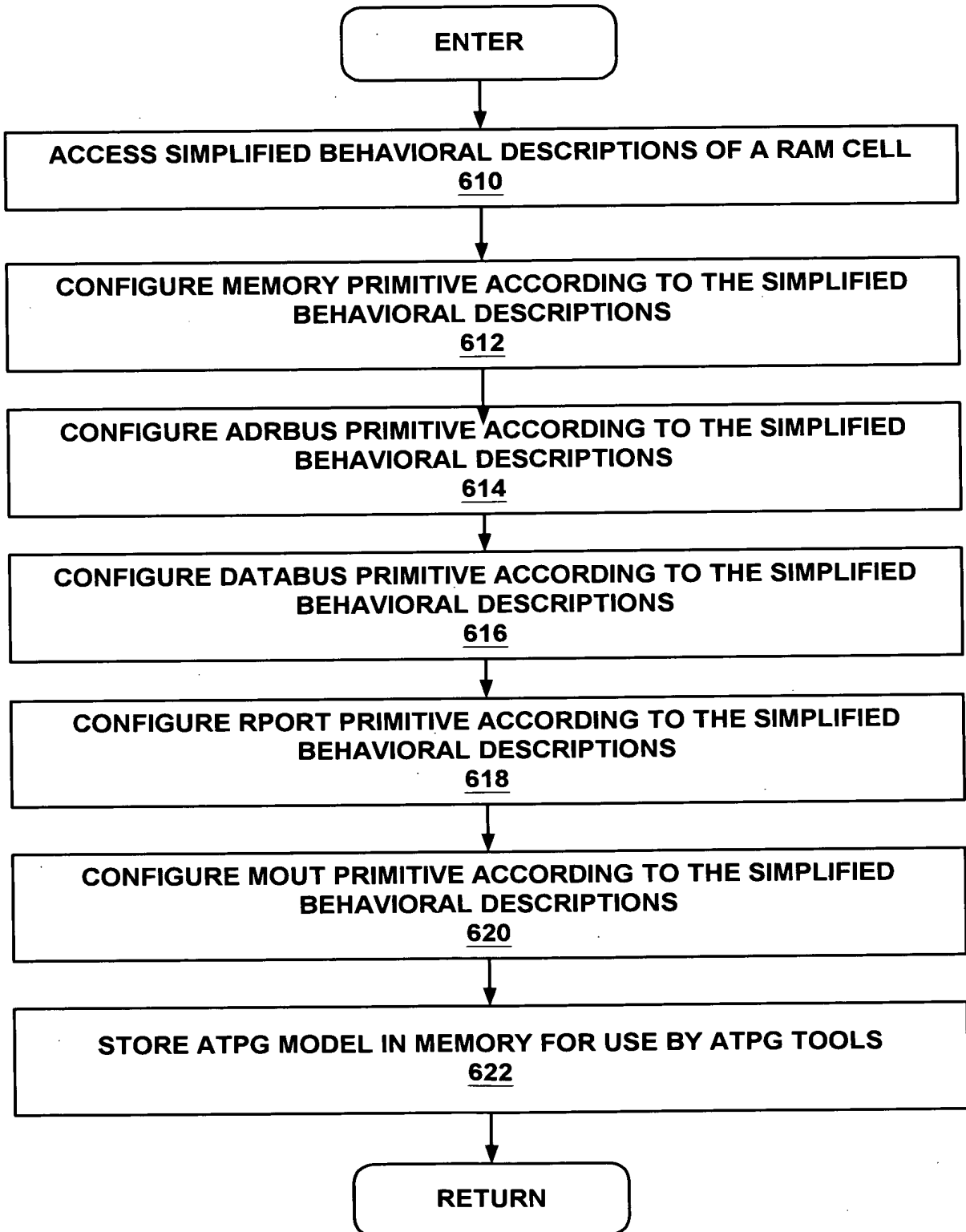
ATPG MEMORY MODEL  
GENERATION PROCESS  
310



**FIGURE 6A**

7/22

310



**FIGURE 6B**

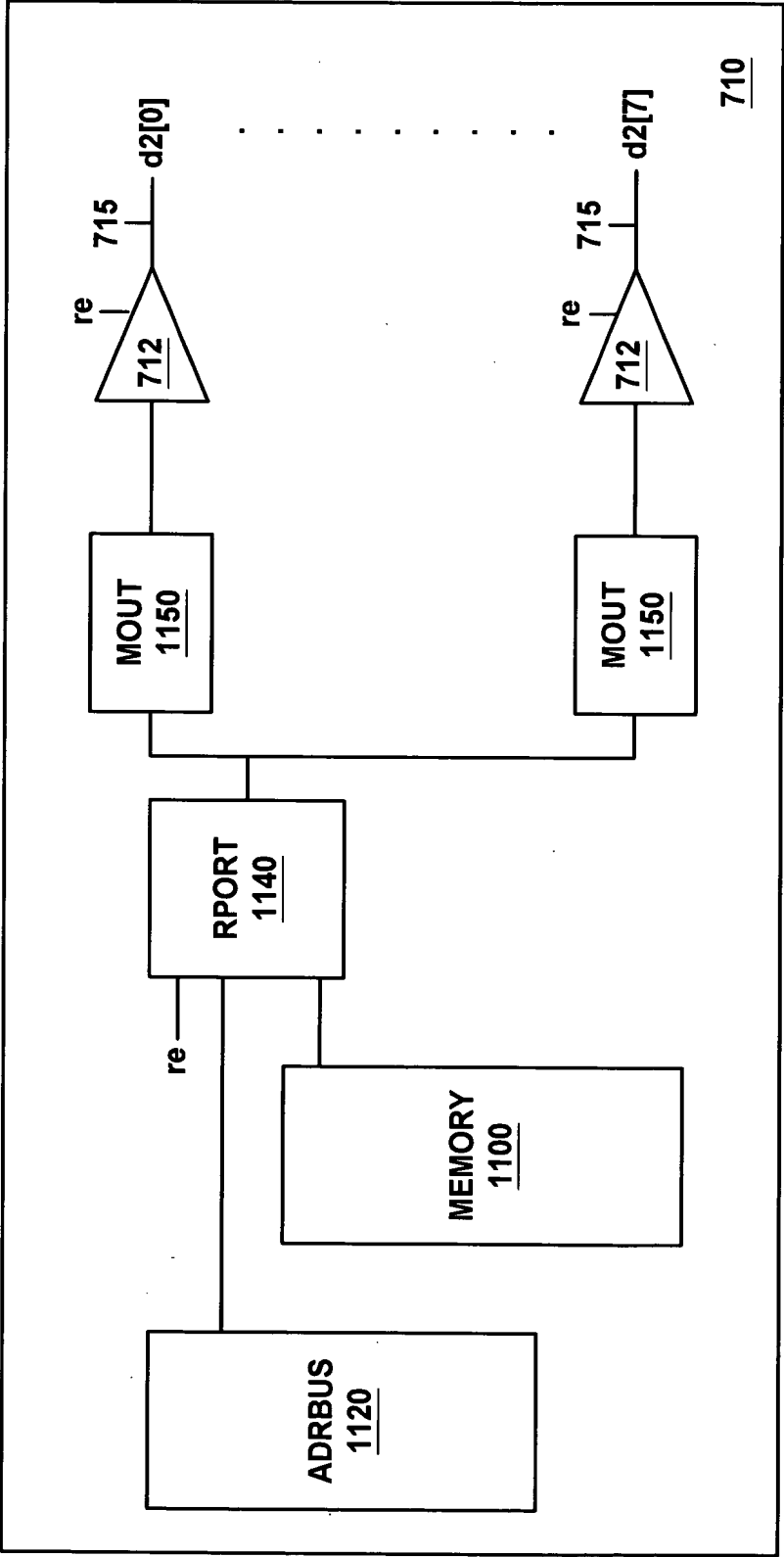
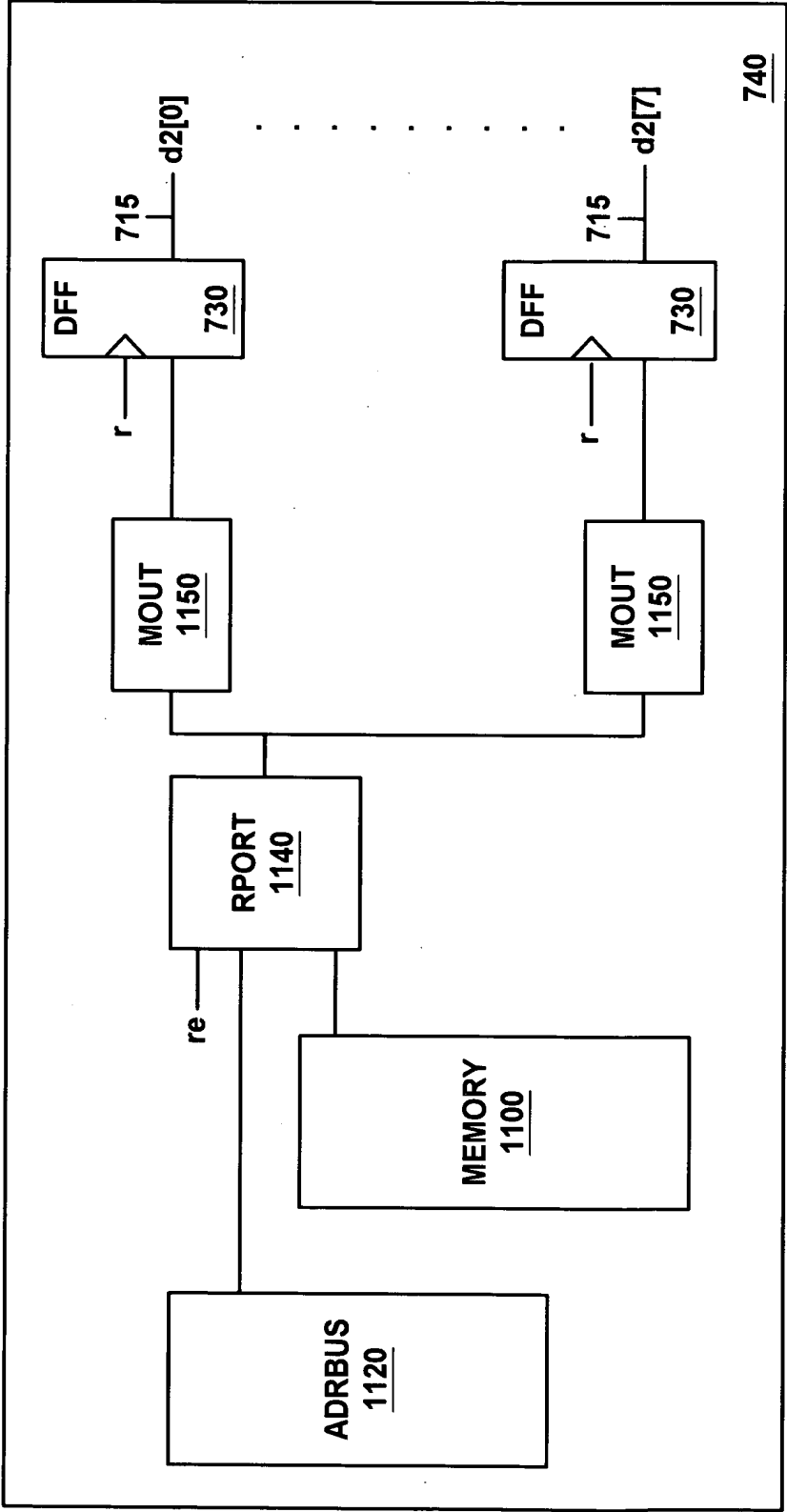


FIGURE 7A





**FIGURE 7B**

# FIGURE 7C

```

module atpgram ( r,w, a, d1, d2 );
input r,w;
input [127:0] a;
input [7:0] d1;
output [7:0] d2;
    reg [7:0] mymem [15:0], d2;
    integer i;
function [6:0] addr_encode;
input [127:0] addr; // decoded address
integer n;
begin
    addr_encode = 7'bx; // init
    for (n=0; n < 128; n=n+1) begin
        if (addr [n] ==1) begin
            addr [n] =0;
            if ( ! addr == 0) addr_encode=n;
            n = 128; // break
        end
    end
end
endfunction

    always @ (posedge w) if (a)
        mymem [addr_encode (a) ] = d1;
    always @ r if (r && a)
        d2 = mymem [addr_encode (a) ];
endmodule

```

**FIGURE 8**

```
input [7:0] comp;
output [3:0] addr;
output hit, mhit;
hit = 0; mhit = 0;
addr = 4'bX;
for (i=0; i<16; i=i+1) begin
  if (mymem[i] == comp) begin
    if (hit==1) mhit = 1;
    hit = 1;
    addr = i; // keep last address
  end
end
```

900**FIGURE 9A**

```
input [7:0] comp, mask;
output [3:0] addr;
output hit, mhit;
hit = 0; mhit = 0;
addr = 4'bX;
for (i=0; i<16; i=i+1) begin
    if ( (mymem [i] & mask) == (comp & mask) )
        begin
            if (hit==1) mhit = 1;
            else addr = i; // keep 1st address
            hit = 1;
        end
    end
end
```

940**FIGURE 9B**

```
input [7:0] comp, mask;
output [15:0] addr;
output hit, mhit;
hit = 0; mhit = 0;
addr = 0;
for (i=0; i<16; i=i+1) begin
    if ( (mymem [i] & mask) ==(comp&mask) )
        begin
            if (hit==1) begin
                hit = 0;
                i = 16; // break;
            end
        end
    else addr [i] = 1; // keep 1st addr
    hit = 1;
end
end
```

970**FIGURE 9C**

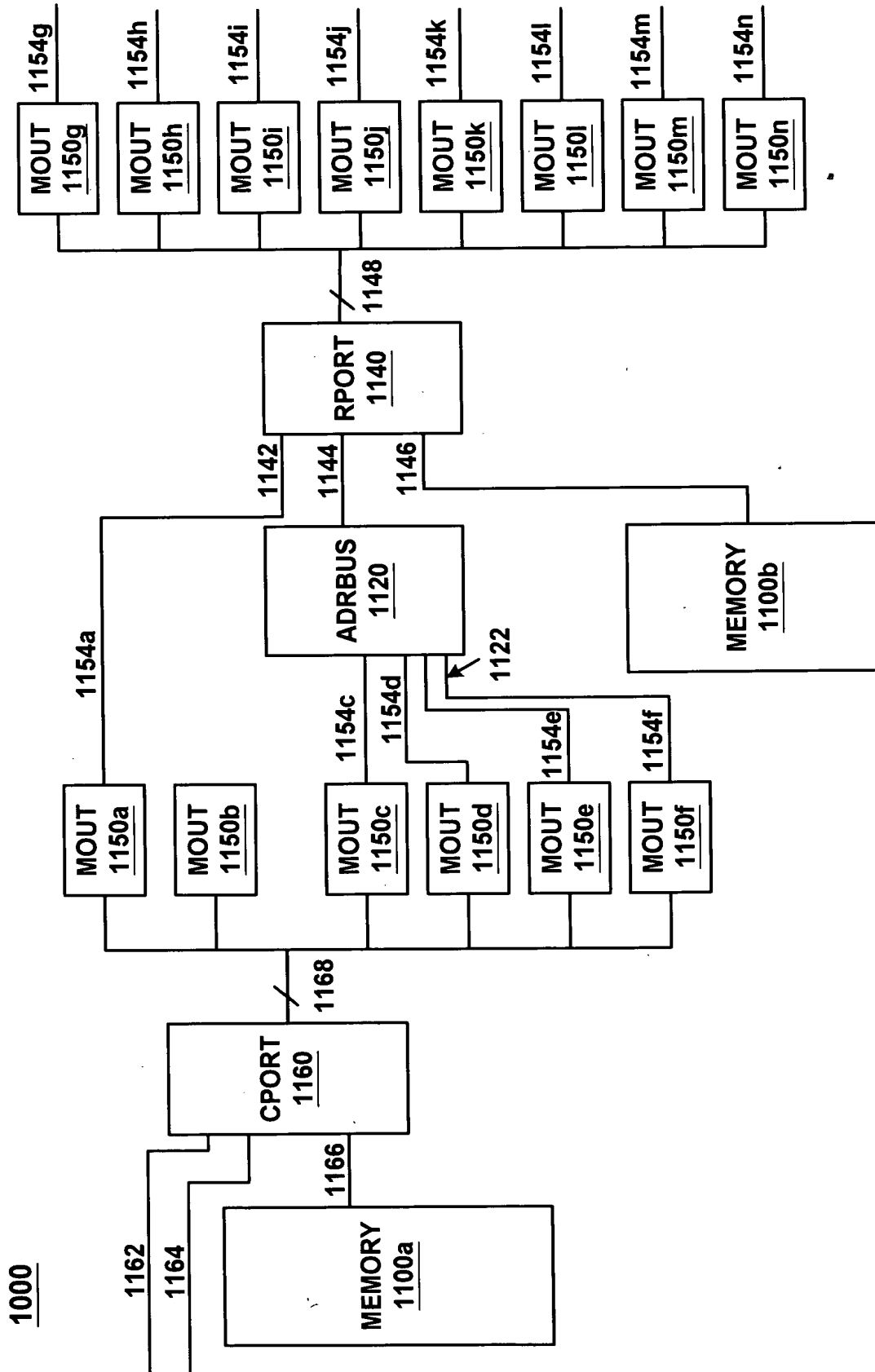
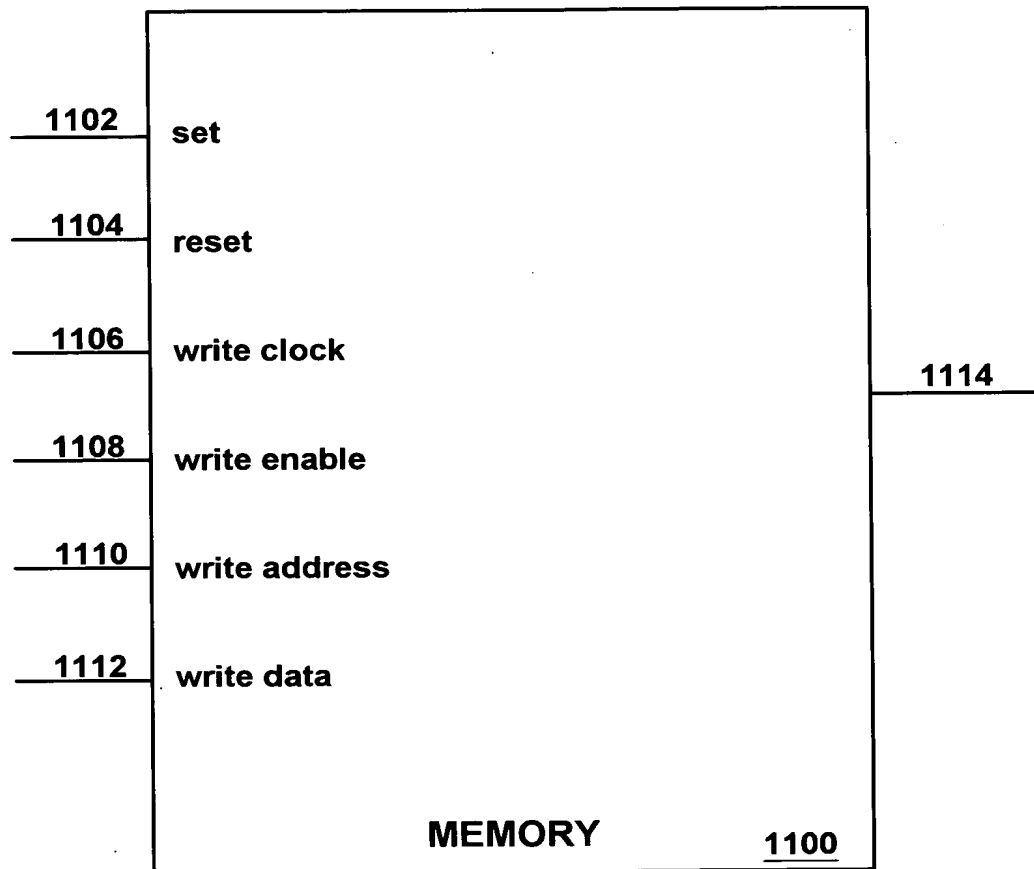
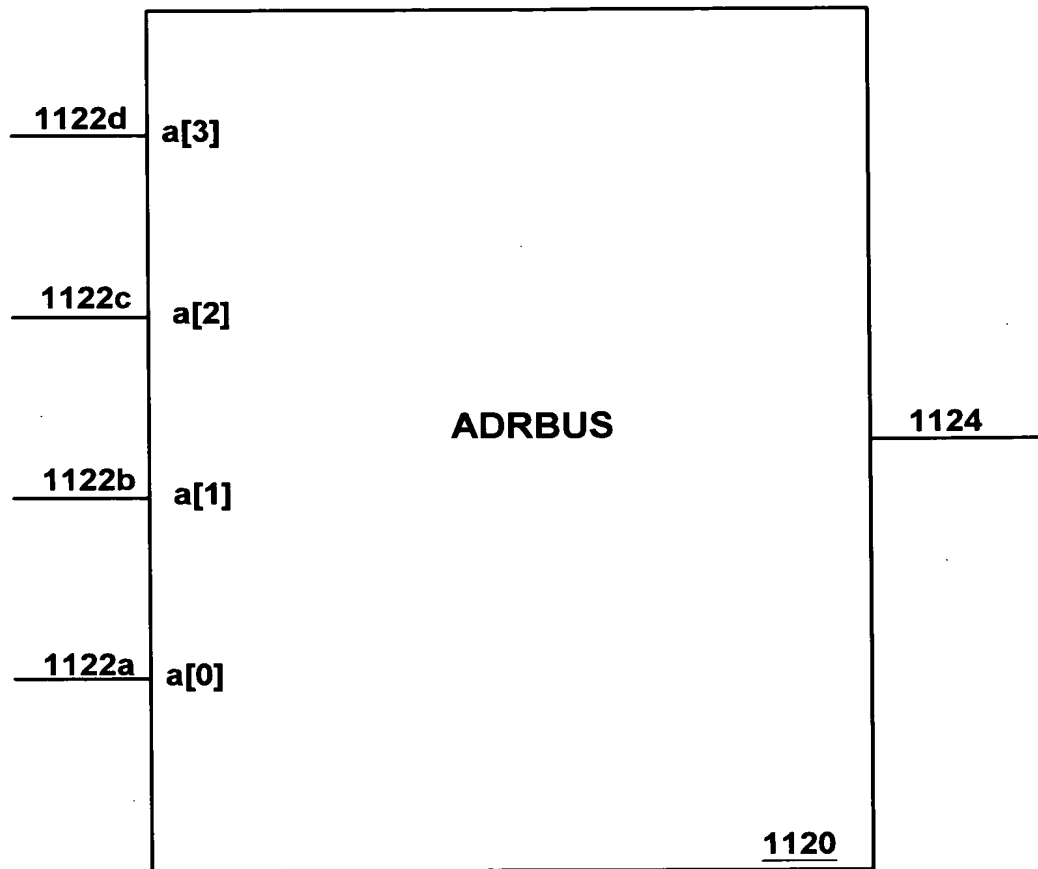
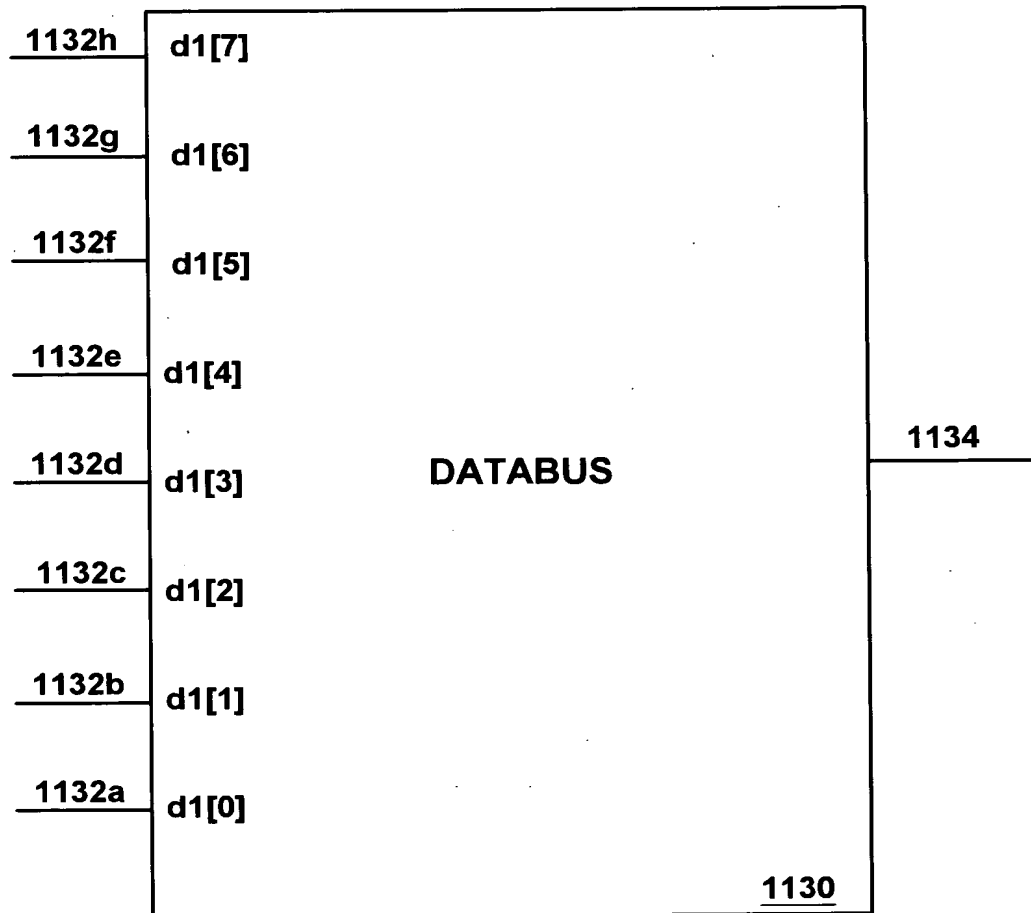


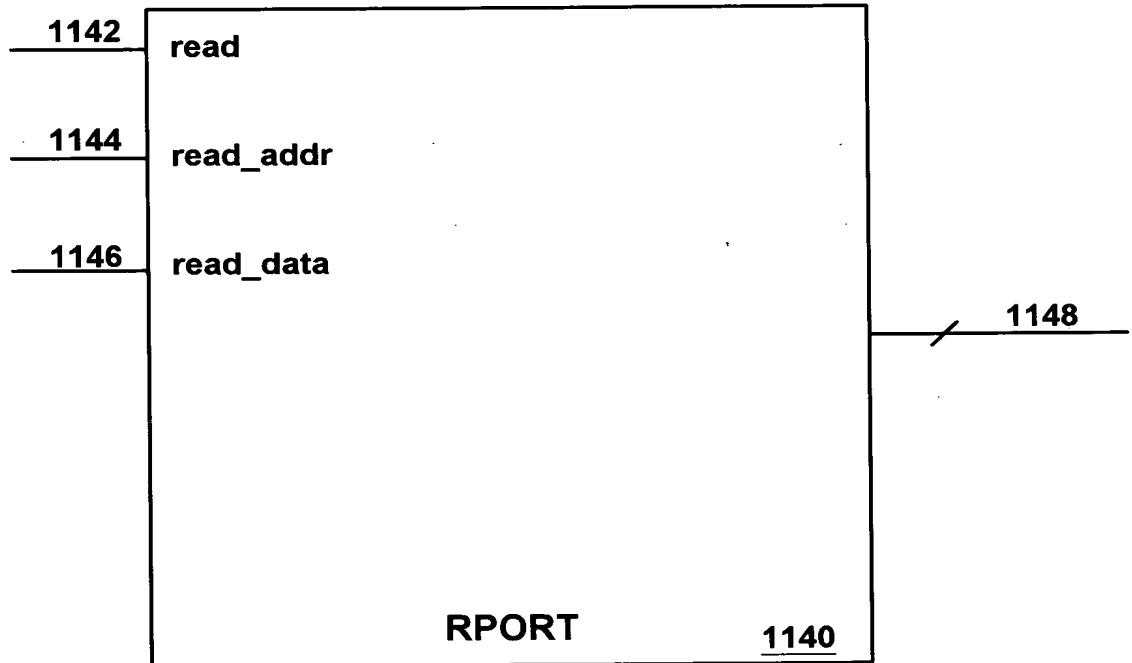
FIGURE 10

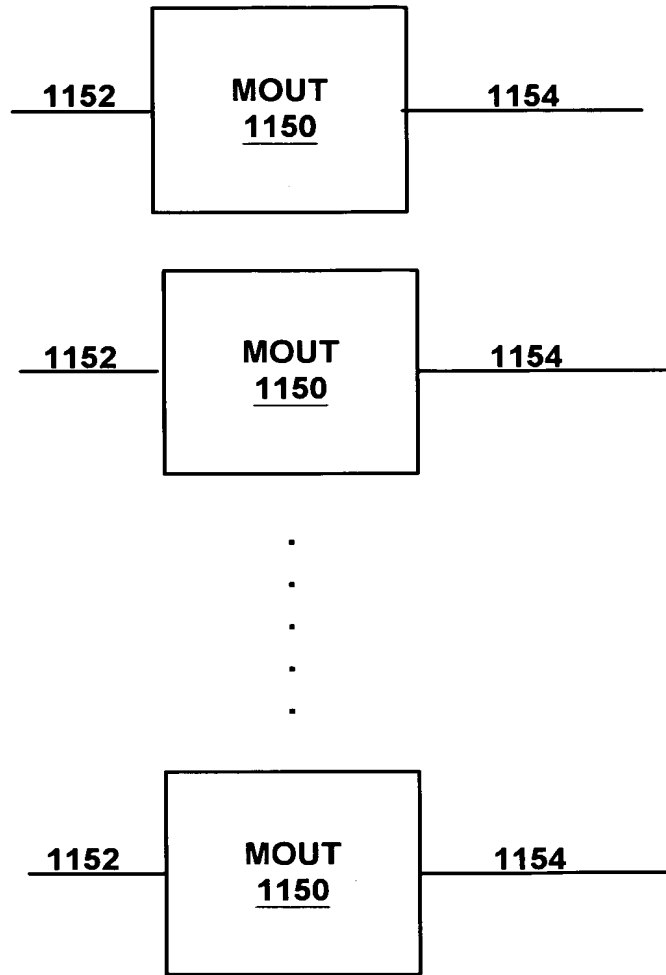
**FIGURE 11A**



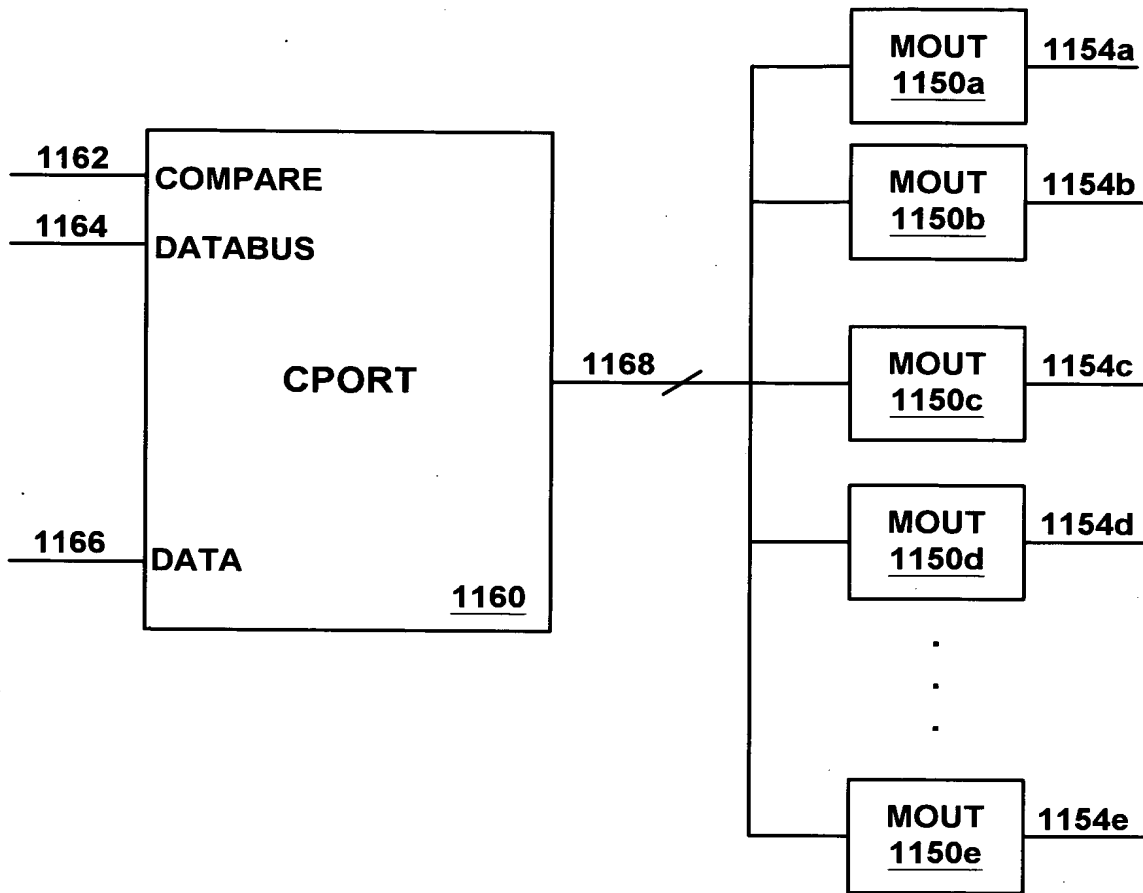
**FIGURE 11B**

**FIGURE 11C**

**FIGURE 11D**



**FIGURE 11E**

**FIGURE 11F**

```
module withram (reset, r,w, a, d1,d2);
input reset, r,w;
input [3:0] a;
input [7:0] d1;
output [7:0] d2;
    reg [7:0] mymem [15:0], d2;
    integer i;
    event mymem_update;
    always @ reset if (reset) begin
        for (i=0; i<16; i=i+1) mymem [i] <=0;
        #0; ->mymem_update; end
    always @ (posedge w) begin
        mymem [a] <= d1; #0; ->mymem_update;
    end
    always @ (r or a or mymem_update)
        if (r) d2 <= mymem [a];
        else d2 <= 8'b0;
endmodule
```

1200**FIGURE 12**